Name: Rami Mohammad Farid Abd-El Hamid

Section: 2

Lexical Scanner Assignment

Introduction

In computer science, lexical analysis is the process of converting a sequence of characters (such as in a computer program or web page) into a sequence of tokens (strings with an identified "meaning"). A program that performs lexical analysis may be called a lexer, tokenizer, or scanner (though "scanner" is also used to refer to the first stage of a lexer). Such a lexer is generally combined with a parser, which together analyze the syntax of programming languages, web pages, and so forth. The assignment was to implement the operation performed by the scanner and read source code from an input file to scan it and divide it into meaningful tokens output in the output file.

Attached is the source file “scanner.cpp”, the output file “scanner_output.txt”, and the input file “tiny_sample_code.txt”.

Code Explanation

The approach of the implementation is as follows: The main function acts as the scanner in which the logic will be used to scan the file.

Enumerated types are used to define a token_type variable which can take the set of values specified in the code. There’s also the char_type enumerated type which define the values a variable of type char_type can take.

A function called get_type(char x) is used to take a character as its input and returns its corresponding enumerated type.

A Token class is used to define a token object. Each token object has two attributes, a category variable of token_type, and lexeme variable of string type.

Along with the attributes, each token object also has two member functions: assign_cat() and reset().

The method assign_cat() is invoked on the token object to give its category attribute a value based on the value in its lexeme attribute.

The reset() member function is invoked on the token_object to reset its attributes values.

Inside main(), we have a vector of tokens, tokens_list, and a temporary token, next_token. Followed by those is the logic for reading and writing from and to the files. The input file is read into a string whole_file.

The main idea of parsing is having a global index by which we index characters in the whole_file string and then have a while loops as long as there are characters left in the file.

We check on the type of the character we are currently looking at and accordingly fit into one of the if statements. Inside each if statement, it's like we jumped to a a state of the DFA; we stay there incrementing the global index and looping on the file again accumulating characters of the same type in the current_lexeme variable till we detect that the current character type has changed, at which case we need to move to the start of the big loop to recheck which if statement we will fit into.

Before leaving each if statement to recheck at the beginning of the big while loop we set the next_token object lexeme attribute to the value we have gathered in out temporary current_lexeme string and invoke the assign_cat() method on the next_token object to give it a token category. After that we push the next_token object to the tokens vector and then reset the next_token object to be ready for another case. We continue doing this until the end of the file taking into consideration the specifications of the language structure.

Lastly, we have a for loop that loops on each token in the tokens_list vector to write the values of each token attributes into the output file.